Süsteemprogrammeerimine keeles C

Loeng 7

Kuidas hoida pakitud faili

- Väike ülevaade probleemistikku

Bittoperaatorid

- Opereerivad numbritel bitikaupa, vaadeldes neid kahendkoodidena (mis õnneks on ka viis, kuidas neid arvutis hoitakse)
- 10ndsüsteemis ei ole nad eriti huvitavad, samas 2nd-, 8nd- ja 16ndsüsteemis on nad veidi sisukamad
- 8ndsüsteemi konstandi tegemiseks kirjutatakse konstandi ette lisanull ja kuueteistkümnendsüsteemi konstandi tegemiseks kirjutatakse ette 0x või 0X
- Nt 077 või 0xff

Oluline!

- Bittidega manipuleerimisel kasutage alati andmetüüpi, mis on **unsigned**, ehk teisisõnu märgita.
- See on oluline bitinihete puhul (allpool)

& = AND

- Üksik ampersand & teeb kahe arvu vahel bitthaaval AND tehet. Iga bitt on 1 siis, kui mõlemas arvus on vastava järgu bitid 1.
- 0x56 & 0x32 = 0x12

```
   01010110
 & 00110010
 ----------
   00010010
```

| = OR

- Püstkriips | teeb kahe arvu vahel bitthaaval OR tehet. Tulemuse vastava järgu bitt on 1 siis, kui ühes või teises arvus on vastavas järgus 1.
- 0x56 | 0x32 = 0x76

```
   01010110
 | 00110010
 ----------
   01110110
```

^ = XOR

- ^ operaator annab tulemuseks kahe arvu omavaheliste bittide XOR tehte. Bitt on 1 siis, kui mõlema arvu vastava järgu bitid on erinevad (1 juhul, kui ainult üks operaatoritest on 1)
- 0x56 ^ 0x32 = 0x64

```
   01010110
 ^ 00110010
 ----------
   01100100
```

~ = täiend

- Tilde ~ operaator annab arvu täiendi (complement). Tehe tehakse ainult ühe arvuga (On unaarne nagu !, & ("address of" tähenduses) ja *). Täiendiks on siis arv, kus on 1 ja 0 vahetatud.
- 16-bitise täisarvu puhul ~0x56 = 0xffa9

  ~0000000001010110

  1111111110101001

<< nihe vasakule

- << operaator nihutab esimese argumendi bitte teise argumendi jagu vasakule, täites uued kohad 0-dega.
- Välja nihkunud bitid heidetakse kõrvale
- 0x56<<2 = 0x58

```
000001010110  <<  2
------------
000101011000
```

>> nihe paremale

- >> operaator nihutab esimest argumenti teise jagu paremale, täites vasakult poolt 0-dega.
- Juhul, kui tegemist on märgiga (*signed*) arvuga ja kõige vasakpoolsem bitt on 1, võib ta sisse nihutada ka 1. (Siit ka eelnev soovitus: bittmuutujate puhul deklareerida muutujad märgita arvudena)
- 0x56 >> 1 = 0x2b

```
01010110  >>  1
--------
00101011
```

Nihete aritmeetiline tähendus

- Nihe vasakule << on sama, mis korrutamine 2-ga
- Analoogselt on 10ndsüsteemis nihe vasakule korrutamine 10-ga
- Nihe paremale >> on jagamine 2-ga, samas võib juhtuda, et säilitatakse kõige vasakpoolsem bitt (st nihutatakse sisse eelmise vasakpoolseimaga identne bitt).
- Nihe on loomulikult kiirem kui korrutamine või jagamine: Enamasti teeb asenduse kompilaator, seega pole tarvis keerukamat koodi kirjutada.

Bittide testimine

- Kontrollimaks, kas mingi bitt on üleval, kasutame me tehet & ja "maski"
- Nt, kui mask on 0x40, siis 0x56 & 0x40 on 0x40, kuid samas 0x32 & 0x40 = 0x00
- Kuna iga nullist erinev väärtus on tõene, võib testimist teha otse:

```
if (x & 0x40) halt_and_catch_fire();
```

1 bitised maskid

- Mõnikord on otstarbekas defineerida hunnik bitimaske, milles on püsti erinev bitt, ning siis mõnd täisarvu *lippude* (*flag*) kogumina käsitleda

```
  #define OVERHEATED 0x01
  #define NOISY 0x02
  #define BEEPING 0x04
  #define WET 0x08
  #define BURNING 0x10
unsigned int flags;
if (flags & NOISY) { /* handle noise */ }
if (flags & BEEPING) { /* handle beep */ }
else { /* handle nonbeeping case */ }
```

- if (flags & NOISY) lugeda: "kui NOISY bitt on püsti".

Bittide määramine

- Bitimaske saab kasutada ka bittide üles lükkamisel. Et BURNING bitt püsti panna, kasutame:

```
flags = flags | BURNING;
flags |= OVERHEATED | BURNING; /* 2 bitti üles */
```

- Biti mahavõtmise loogika: jätame üles kõik bitid, v.a. mahavõetav bitt. Teeme seda ~ ja & tehetega:

```
flags = flags & ~BURNING;
flags &=  ~(OVERHEATED|BURNING);
```

- Bitti saab ^ abil ümber lülitada (st vahetada olekut):

```
flags = flags ^ BURNING;
flags ^= BURNING;
```

Bitiväljad (*Bit fields*)

- Alternatiiviks eelnevale pakub C bitiväljasid, mis on viis väljade poole otse pöördumiseks. Bitiväli on komplekt lähestikku paiknevaid bitte mingis defineeritud muutujas.

```
struct {
   unsigned int is_keyword: 1;
   unsigned int is_extern: 1;
   unsigned int is_static: 1;
} flags;

flags.is_static = 1;
flags.is_keyword = 0;
```

Bitiväljad (2)

- Bitiväljad deklareeritakse märgita integeridena, et neil ei tekiks märgist lähtuvaid probleeme. Väljade poole pöördutakse nagu struktuuri liikmete poole.
- Nad teisendatakse iga tehte puhul täisarvuks.
- Bitiväljade siseehitus sõltub palju konkreetsest implementatsioonist.

Varieeruva argumentide arvuga funktsioonid

- Varieeruva argumentide arvuga funktsioonide deklareerimiseks tuleb funktsiooni argumendid deklareerida nii, nagu me ei viitsiks neid kõiki välja kirjutada:

```
void foo( int arg1, int arg2, ...) {
```

- Kuidas neid argumente kätte saada?

Ülejäänud argumentide kättesaamine

Näide sehkendatakse kuskile teise kohta

Ülejäänud argumentide kättesaamine (lihtsam meetod)

- #include <stdarg.h>
- Failis on makrod/funktsioonid:

```
void va_start(va_list ap, last);
type va_arg(va_list ap, type);
void *va_end(va_list ap);
```

- Üks võimalik implementatsioon:

```
typedef char *va_list;
#define va_start(ap, v)  ((void) (ap = (va_list) &v + sizeof(v)))
#define va_arg(ap,type)   (*((type *)(ap))++)
#define va_end(ap)  ((void) (ap = 0))
```

va_start() initsialiseerib argumentide nimistu, last ei tohiks olla registrimuutuja, massiiv ega funktsioon.
va_arg() annab järgmise etteantud tüübist argumendi
va_end() teeb vajadusel puhastust

- Lisainfo: 'man stdarg' või 'man vararg'

Varieeruv argumentide arv (näide)

```
#include <stdarg.h>
#define MAXARGS     31
void f1(int n_ptrs, ...) {
    va_list ap;
    char *array[MAXARGS];
    int ptr_no = 0;

    if (n_ptrs > MAXARGS)
        n_ptrs = MAXARGS;
    va_start(ap, n_ptrs);
    while (ptr_no < n_ptrs)
        array[ptr_no++] = va_arg(ap, char*);
    va_end(ap);
    f2(n_ptrs, array);
}
```

string.h

- Üldinfo
  - stringilibra funktsioonid arvestavad, et \0 lõpetab stringi
  - kui \0 puudub, võib väljund olla "natuke" vale

```
char dst[100], s[]="abc";
s[3] = 'd'; /* '\0' kirjutatakse üle */
strcpy(dst, s); /* oih! */
```

Stringi pikkus

- size_t strlen(const char *s);
  - tagastab stringi pikkuse
  - const sellises deklaratsioonis sümboliseerib, et s ei ole funktsioonisiseselt muudetav
  - size_t on enamasti kas unsigned int või unsigned long

```
size_t strlen(const char *s) {
 size_t n;
 for (n = 0; *s != '\0'; s++)
    n++;
 return n;
}
```

```
size_t strlen(const char *s) {
    const char *t = s;
    while (*t)
       t++;
    return t - s;
}
```

Stringi kopeerimine

- char *strcpy(char *dest, const char *src);
  - kopeerib stringi *src* stringiks *dest*
  - jälgida, et *dest* oleks *src* mahutamiseks piisavalt suur
- char *strncpy(char *dest, const char *src, size_t n);
  - kopeeritakse maksimaalselt n baiti
  - kui on täpselt n, siis \0 jääb lõpust ära
  - kui on vähem kui n, kirjutatakse ülejäänutesse \0

Stringide võrdlemine

- int strcmp(const char *s1, const char *s2);
  - võrdleb stringe omavahel
  - tagastab 0, kui stringid on võrdsed
  - kui s1 > s2, siis tagastab arvu , mis on suurem kui 0
  - ja vastupidi

```
"str1"   >  "str0"
"str1"   == "str1"
"str1"   <  "str2"
```

  - levinud viga:

```
if ( strcmp(a, b) ) ... // VIGA!
```

- int strncmp(const char *s1, const char *s2, size_t n)

Stringide jupitamine

- char *strtok(char *str, const char *delim);
  - tagastab stringi *str* järgmise jupi või NULL, kui neid enam pole
  - argumendina esimeses väljakutses jagatav string, edaspidi NULL
  - delim on string juppe eraldavatest charidest
  - puudusteks on see, et me ei saa teada, mis see eraldusmärk oli ja see, et esimest argumenti muudetakse

strtok() näide

```
void print_tokens(char *line)
{
  char whitespace[]=" \t\f\r\v\n";
  char *token;
  for (token = strtok(line, whitespace); token != NULL;
          token=strtok(NULL, whitespace)) // NULL!
   printf("Järgmine on %s\n", token);
}
```

Ülevaade stringifunktsioonidest (1)

- **char *strcat(s, cs)** Liidab stringi s otsa stringi cs ja tagastab s-i
- **char *strncat(s,cs, n)** Lisab stringi s otsa maksimaalselt n stringi cs märki
- **char *strcpy(s, cs)** Kopeerib cs-i s-iks, ka \0
- **char *strncpy(s,cs)** Kopeerib cs-i maksimaalselt n tähemärki s-iks, ülejäänud määrab \0-deks
- **char *strtok(s,cs)** Leiab stringi s seest cs-i märkidega eraldatud osad
- **size_t strlen(cs)** Tagastab cs-i pikkuse (v.a. \0)
- **int strcmp(cs1,cs2)** Võrdleb cs1 ja cs2, tagastab nullist suurema, võrdse või väiksema arvu vastavalt sellele, kas cs1 on cs2-st vastavalt suurem, võrde või väiksem
- **int strncmp(cs1, cs2, n)** Võrdleb cs1 ja cs2 esimesi n-i tähte omavahel

Ülevaade stringifunktsioonidest (2)

- **char *strchr(cs,c)** Annab esimese c esinemise peale stringis s pointeri
- **char strrchr(cs,c)** Annab viimase c esinemise peale stringis s pointeri
- **char *strpbrk(cs1, cs2)** Otsib stringist cs1 esimese cs2 tähe esinemise ja annab sellele pointeri
  - nt: strpbrk(“Selle lause lõpp on kirjas maja peal aadressil”, “cba”) annab tagasi pointeri tähele a sõnas "lause"..
- **char strstr(cs1, cs2)** Otsib stringist cs1 alamstringi cs2
  - Eelmised 4 annavad ebaedu korral kõik tagasi NULL
- **size_t strspn(cs1, cs2)** Tagastab cs1 algusest pikkuse, mis koosneb ainult cs2-s olevatest tähtedest.
  - nt:. strspn("Siil udus", “ilu S”) == 6
- **size_t strcspn(cs1, cs2)** Tagastab cs1 algusest pikkuse, milles ei esine cs2 tähti
  - nt strcspn("Siil udus", "abs") == 8

string.h failist leiate neid veel!

strpbrk()

- char *strpbrk(const char *s, const char *accept);
  - pbrk – pointer to break

```
char string[100] = "The 3 men and 2 boys ate 5 pigs\n";
 char *result;
 printf( "1: %s\n", string );
 result = strpbrk( string, "0123456789" );
 printf( "2: %s\n", result++ );
 result = strpbrk( result, "0123456789" );
 printf( "3: %s\n", result++ );
 result = strpbrk( result, "0123456789" );
 printf( "4: %s\n", result );
```

strspn()

- size_t strspn(const char *s, const char *accept);
  - ütleb mitu tähte stringist koosneb täielikult *accept*-i tähtedest

```
char string[] = "cabbage";
int  result;
result = strspn( string, "abc" );
printf( "The portion of '%s' containing only "
 "a, b, or c is %d bytes long\n", string, result );
```

strstr()

- char *strstr(const char *haystack,
  const char *needle);

```
char str[] =    "lazy";
char string[] = "The quick brown dog jumps over the lazy
   fox";
char fmt1[] =   "         1         2         3         4
      5";
char fmt2[] =
   "12345678901234567890123456789012345678901234567890";
char *pdest;    int  result;

printf( "String to be searched:\n\t%s\n", string );
printf( "\t%s\n\t%s\n\n", fmt1, fmt2 );
pdest = strstr( string, str );
result = pdest - string + 1;
if( pdest != NULL )
   printf( "%s found at position %d\n\n", str, result );
else
   printf( "%s not found\n", str );
}
```

Valge slaid